United States Patent Application

of

Ann M. Wollrath, James H. Waldo, Peter C. Jones, Kenneth Arnold

for

# A METHOD AND APPARATUS FOR TRANSPORTING BEHAVIOR IN AN EVENT-BASED DISTRIBUTED SYSTEM

Related Applications

The following identified U.S. patent applications are relied upon and are incorporated by reference in this application.

Provisional U.S. Patent Application No. _____, entitled "Distributed Computing System," filed on February 26, 1998.

U.S. Patent Application No. _____, entitled "Method and System for Leasing Storage," bearing attorney docket no. 06502.0011-01000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method, Apparatus, and Product for Leasing of Delegation Certificates in a Distributed System," bearing attorney docket no. 06502.0011-02000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method, Apparatus and Product for Leasing of Group Membership in a Distributed System," bearing attorney docket no. 06502.0011-03000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Leasing for Failure Detection," bearing attorney docket no. 06502.0011-04000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Deferred Reconstruction of Objects and Remote Loading for Event Notification in a Distributed System," bearing attorney docket no. 06502.0062-01000, and filed on the same date herewith.

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N.W.
WASHINGTON, DC 20005
202-408-4000

INS A2 Cont

U.S. Patent Application No. _____, entitled "Methods and Apparatus for Remote Method Invocation," bearing attorney docket no. 06502.0102-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and System for Deterministic Hashes to Identify Remote Methods," bearing attorney docket no. 06502.0103-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," bearing attorney docket no. 06502.0104-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Downloadable Smart Proxies for Performing Processing Associated with a Remote Procedure Call in a Distributed System," bearing attorney docket no. 06502.0105-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Suspension and Continuation of Remote Methods," bearing attorney docket no. 06502.0106-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and System for Multi-Entry and Multi-Template Matching in a Database," bearing attorney docket no. 06502.0107-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and System for In-Place Modifications in a Database," bearing attorney docket no. 06502.0108, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and System for Typesafe Attribute Matching in a Database," bearing attorney docket no. 06502.0109-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Method and System for Facilitating Access to a Lookup Service," bearing attorney docket no. 06502.0113-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Apparatus and Method for Dynamically Verifying Information in a Distributed System," bearing attorney docket no. 06502.0114-00000, and filed on the same date herewith.

U.S. Patent Application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on February 26, 1998.

U.S. Patent Application No. _____, entitled "An Interactive Design Tool for Persistent Shared Memory Spaces," bearing attorney docket no. 06502.0116-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Polymorphic Token-Based Control," bearing attorney docket no. 06502.0117-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Stack-Based Access Control," bearing attorney docket no. 06502.0118-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Stack-Based Security Requirements," bearing attorney docket no. 06502.0119-00000, and filed on the same date herewith.

U.S. Patent Application No. _____, entitled "Per-Method Designation of Security Requirements," bearing attorney docket no. 06502.0120-00000, and filed on the same date herewith.

Field of the Invention

This invention relates generally to a distributed computer system and more specifically to event handling procedures in a distributed computer system.

Background of the Invention

Distributed computer systems are systems in which the programming and data that the computer operates on are spread out over more than one computer, usually over a network with many computers. One conventional method for organizing a distributed system uses the client-server model, in which one part of the distributed system (the client) asks for some service from another part of the distributed system (the server). The server responds and then handles the client's request appropriately.

Under the client-server model, when the client encounters a procedure located at the server, the procedure may be implemented using some form of a Remote Procedure Call (RPC): the client informs the server that it would like the procedure executed using certain parameters, the server executes the procedure, and the server returns the results of the execution to the client.

An alternative method for organizing distributed systems uses the event/notification model. In this model, a "listener" process, which is interested in the occurrence of an event within another entity within the system, registers its interest with a second process, a "notifier" process, designed to monitor such events. When the event occurs, the notifier notifies the listener or notifies another process (i.e., a third party process) designated by the listener when registering with the notifier. After receiving the event notification, that notified entity (i.e., the listener or the third party listener, as the case may be) may execute a function designated in the

notification and present in the address space of the notified process. In the event-notification model, there is no single point of control as in the client-server model; instead, control is determined by notifications sent in response to the occurrence of designated events.

In a non-distributed system, the association of an event with the function to be run in response to the notification, called a callback function, is straightforward, since the event, the listener, and the callback function are all in a single address space. Life is not so simple in a distributed system. The listener may be in one address space or physical machine, the event may be generated in a second address space or physical machine (e.g., the address space of the notifier or an address space that the notifier monitors), and the notification may be sent to a third address space or physical machine (e.g., the address space of the third party process). This makes it difficult for the listener to insure that the third party process will be able to respond to the event in an appropriate manner, as the two may be separated and, possibly, might not even know of the other's existence. For example, the listener, to ensure that the correct function is executed by the third party, must keep track of whether the function is available in the third party's address space. This can be particularly burdensome when an administrator desires to update functions in the third party's address space, as all the potential distributed users of the new function must be updated on the status of the new function.

Thus, there is a need in the art to more effectively handle distributed programs collaborating using the event-notification model. This need is particularly poignant in systems in which the requesting entity is not able to easily verify the status of a function present at the executing entity.

Summary of the Invention

Objects and advantages of the invention will be set forth in part in the description which follows, and in part will be obvious from the description, or may be learned by practice of the invention. The objects and advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims.

To achieve the objects and in accordance with the purpose of the invention, as embodied and broadly described herein, a first aspect consistent with the present invention includes a method for controlling program execution in a distributed computer system comprising the steps of: (1) registering interest in an occurrence of an event in the distributed computer system, the registration of interest including information identifying the occurrence of the event, an identifier of a software entity in the distributed system, and a first object including a process and parameter data corresponding to the process; (2) monitoring at least a portion of the distributed computer system for the occurrence of the registered event; (3) notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred; and (4) executing methods contained within the first object in response to the notifying step.

To achieve the objects and in accordance with the purpose of the invention, as embodied and broadly described herein, a second aspect consistent with the present invention includes a protocol for controlling the execution of processes in a distributed computer system, the protocol comprises a number of steps, including: (1) receiving a registration of interest in an event that is expected to occur in the distributed computer system, the registration including an identifier of a software entity in the distributed system and a first object, the first object including computer

instructions for performing a process and parameter data corresponding to the process; (2) monitoring the distributed system for the occurrence of the registered event; and (3) notifying the software entity identified in the registration of interest when the event occurs, the notification including a copy of the first object and an identification of the event that occurred.

A third aspect of the present invention is a computer system comprising memories having first, second, and third virtual machines, respectively. The second virtual machine executes a process that receives, from the first virtual machine, a registration of interest in an event and transmits a message in response to the occurrence of the event, the registration of interest and the message including an object., The third virtual machine receives the message and executes methods contained within the object.

Additional aspects of the present invention, related to the first aspect, are directed to a computer readable medium and a computer data signal embodied in a carrier wave.

Brief Description of the Drawings

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments consistent with this invention and, together with the description, help explain the principles of the invention. In the drawings,

Fig. 1 is a diagram of an exemplary distributed system;

Fig. 2 is a diagram of an exemplary computer within the exemplary distributed system;

Fig. 3A is a flow chart illustrating methods consistent with the present invention for notifying and transporting behavior in an event based distributed system;

Fig. 3B is a table showing exemplary event notification requests stored in a computer memory; and

Fig. 4 is a block diagram illustrating a distributed computing system containing three exemplary computer platforms connected to one another via a network.

Detailed Description

This disclosure describes a protocol allowing a listener process to register interest in an event occurring in another address space or physical machine in such a way as to allow the subsequent notification of the event's occurrence to contain an object that may include methods that are to be run on receipt of the notification. When the notification is received, either by the listener or by a third party, the methods may be executed as specified by the listener.

Referring to the accompanying drawings, a detailed description of an embodiment consistent with the present invention will now be described.

Methods and systems consistent with the present invention operate in a distributed system ("the exemplary distributed system") with various components, including both hardware and software. The exemplary distributed system (1) allows users of the system to share services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system. To accomplish these goals, the exemplary distributed system utilizes the Java™ programming environment to allow both code and data to be moved from device to device in a seamless manner. Accordingly, the exemplary distributed system is layered on top of the Java programming environment and exploits the

characteristics of this environment, including the security offered by it and the strong typing provided by it. The Java programming environment is more clearly described in Jaworski, Java 1.1 Developer's Guide, Sams.net (1997), which is incorporated herein by reference.

In the exemplary distributed system, different computers and devices are federated into what appears to the user to be a single system. By appearing as a single system, the exemplary distributed system provides the simplicity of access and the power of sharing that can be provided by a single system without giving up the flexibility and personalized response of a personal computer or workstation. The exemplary distributed system may contain thousands of devices operated by users who are geographically disperse, but who agree on basic notions of trust, administration, and policy.

Within the exemplary distributed system are various logical groupings of services provided by one or more devices, and each such logical grouping is known as a Djinn. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

Both users and devices may join a Djinn. When joining a Djinn, the user or device adds zero or more services to the Djinn and may access, subject to security constraints, any one of the services it contains. Thus, devices and users federate into a Djinn to share access to its services. The services of the Djinn appear programmatically as objects of the Java programming environment, which may include other objects, software components written in different

programming languages, or hardware devices. A service has an interface defining the operations that can be requested of that service, and the type of the service determines the interfaces that make up that service.

Fig. 1 depicts the exemplary distributed system 100 containing a computer 102, a computer 104, and a device 106 interconnected by a network 108. The device 106 may be any of a number of devices, such as a printer, fax machine, storage device, computer, or other devices. The network 108 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising the exemplary distributed system 100, one skilled in the art will appreciate that the exemplary distributed system 100 may include additional computers or devices.

Fig. 2 depicts the computer 102 in greater detail to show a number of the software components of the exemplary distributed system 100. One skilled in the art will appreciate that computer 104 or device 106 may be similarly configured. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. The memory 202 includes a lookup service 212, a discovery server 214, and a Java™ runtime system 216. The Java runtime system 216 includes the Java™ remote method invocation system (RMI) 218 and a Java™ virtual machine 220. The secondary storage device 204 includes a JavaSpace™ 222.

As mentioned above, the exemplary distributed system 100 is based on the Java programming environment and thus makes use of the Java runtime system 216. The Java runtime system 216 includes the Java™ API, allowing programs running on top of the Java runtime system to access, in a platform-independent manner, various system functions, including

windowing capabilities and networking capabilities of the host operating system. Since the Java API provides a single common API across all operating systems to which the Java runtime system 216 is ported, the programs running on top of a Java runtime system run in a platform-independent manner, regardless of the operating system or hardware configuration of the host platform. The Java runtime system 216 is provided as part of the Java™ software development kit available from Sun Microsystems of Mountain View, CA.

The Java virtual machine 220 also facilitates platform independence. The Java virtual machine 220 acts like an abstract computing machine, receiving instructions from programs in the form of byte codes and interpreting these byte codes by dynamically converting them into a form for execution, such as object code, and executing them. RMI 218 facilitates remote method invocation by allowing objects executing on one computer or device to invoke methods of an object on another computer or device. Both RMI and the Java virtual machine are also provided as part of the Java software development kit.

The lookup service 212 defines the services that are available for a particular Djinn. That is, there may be more than one Djinn and, consequently, more than one lookup service within the exemplary distributed system 100. The lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. The lookup service 212 and its access are described in greater detail in co-pending U.S. Patent Application No. _____, entitled "Method and System for Facilitating Access to a Lookup Service," which has previously been incorporated by reference.

The discovery server 214 detects when a new device is added to the exemplary distributed system 100, during a process known as boot and join or discovery, and when such a

new device is detected, the discovery server passes a reference to the lookup service 212 to the new device, so that the new device may register its services with the lookup service and become a member of the Djinn. After registration, the new device becomes a member of the Djinn, and as a result, it may access all the services contained in the lookup service 212. The process of boot and join is described in greater detail in co-pending U.S. Patent Application No. _____, entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System," which has previously been incorporated by reference.

The JavaSpace 222 is an object repository used by programs within the exemplary distributed system 100 to store objects. Programs use the JavaSpace 222 to store objects persistently as well as to make them accessible to other devices within the exemplary distributed system. JavaSpaces are described in greater detail in co-pending U.S. Patent Application No. 08/971,529, entitled "Database System Employing Polymorphic Entry and Entry Matching," assigned to a common assignee, filed on November 17, 1997, which is incorporated herein by reference. One skilled in the art will appreciate that the exemplary distributed system 100 may contain many lookup services, discovery servers, and JavaSpaces.

Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage

devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the SunLogo, Java, and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.

Fig. 3A is a flow chart illustrating methods consistent with the present invention for notifying and transporting behavior in an event-based distributed system. Listener objects interested in the occurrence of a certain event must register an interest in the event. In particular, for each interest to be registered, the registering object sends a message to a notifier process, which is preferably an object containing procedures for monitoring the occurrence of the event (step 301). Generally, the notifier will be present in the virtual machine in which the event is expected to occur. The registration message includes: (1) implicit or explicit information identifying the event that is to be monitored, (2) information identifying the object that is to be notified when the event occurs, and (3) an object, or a reference to an object, that is to be passed to the notified object when the event occurs. The notifier may use a table to store information relating to registration messages it has received, such as the table shown in Fig. 3B. The notifier monitors its system or the network for the occurrence of the event (step 302). When the event occurs (step 303), the notifier notifies the object identified by the registering object that the registered event has occurred (step 304). Consistent with the present invention, the notification includes an identification of the event that occurred and the object or reference to the object that was passed to the notified object when the event occurred (step 304).

An "event" as used throughout this specification can be broadly defined as a change of system state. An event may be, for example, a timer event, a mouse click event, or a disk access event.

Fig. 4 is a block diagram illustrating a distributed computing system containing three exemplary computer systems 412, 413, and 414, connected to one another via a network. Computer systems 412-414 are physically similar to computer systems 102 and 112. In particular, each of computer systems 412, 413, and 414 includes a computer platform or machine 409, 410, and 411, respectively, executing a virtual machine 406, 407, or 408. Processes 402, 403, and 404 reside on their respective virtual machines 406, 407, and 408. Consistent with the present invention, computer systems 412, 413, and 414 form a distributed computing system using the event/notification model.

A hypothetical situation illustrating an application of the present invention will now be described with reference to the distributed system shown in Fig. 4. Assume that "administrator" process 402, executing in virtual machine 406, has the responsibility of monitoring the network and reporting potential problems to a human operator--such as a disk drive running out of free space. Notifier process 403, executing on virtual machine 407, monitors the system for the occurrence of disk drive full events. To register interest in a "DiskFull" event, process 402 transmits a registration message, called, for example, "RegisterInterest," to notify process 403, such as the message:

RegisterInterest (DiskFull, ReferenceToVM408, SendPage("<parameters> ")).

In this message, "DiskFull" identifies the event that is to be monitored, "ReferenceToVM408" is a reference to the entity that is to be notified in response to the occurrence of the event (in this

case, virtual machine 408), and "SendPage" is an object that includes methods and data that, when executed on a computer having a telephone connection and the ability to dial a pager number, will instruct the computer to page a person at a specified number.

When the DiskFull event occurs, notifier process 403 notifies the designated entity, virtual machine 408, using a generic notify method including the SendPage object, or a reference to the "SendPage" object, and an indication that the DiskFull event has occurred. In response, virtual machine 408 may create a process such as process 404 that executes the methods designated in the SendPage object. The SendPage object will then cause computer system 411 to page the appropriate technician.

As mentioned above, the notify method is preferably a generic method, meaning each virtual machine and/or computer platform may include its own unique implementation of the notify method, although each implementation must meet the specifications required by the definition of the generic notify method.

Because the SendPage object includes, within itself, all required methods and data for execution, SendPage is considered to be an object having "closure." By passing an object having closure to a designated entity, the object initially registering the object does not have to be aware of the functions available at the final destination of the passed object. This is particularly advantageous as it allows the object initially registering the event to easily modify the functionality of the passed object. For example, if the policy (functionality) of the SendPage object is created by an operator at computer 412, it is relatively easy for the operator to modify the policy of SendPage by, for example, instructing SendPage to page a backup technician if the page is not returned by the primary technician within twenty minutes.

Although SendPage was described as having closure by virtue of having all its required methods and data encapsulated within the object, other objects may be passed that have closure even though code still must be supplied by the executing system. In particular, if the code that must be supplied by the executing system implements well known methods that can be implemented in any one of a number of acceptable ways, the object may have closure merely by referencing the generic method definitions and allow the executing system to provide the precise implementation.

While there has been illustrated and described what are at present considered to be preferred embodiments and methods of the present invention, it will be understood by those skilled in the art that various changes and modifications may be made, and equivalents may be substituted for elements thereof without departing from the true scope of the invention. For example, while the foregoing systems and methods have been described with reference to a Java based run-time environment, other run-time environments could conceivably be used to implement the present invention.

In addition, many modifications may be made to adapt a particular element, technique or implementation to the teachings of the present invention without departing from the central scope of the invention. Therefore, it is intended that this invention not be limited to the particular embodiments and methods disclosed herein, but that the invention include all embodiments falling within the scope of the appended claims.